

Exploiting Budan-Fourier and Vincent's Theorems for Ray Tracing 3D Bézier Curves

Alexander Reshetov
NVIDIA

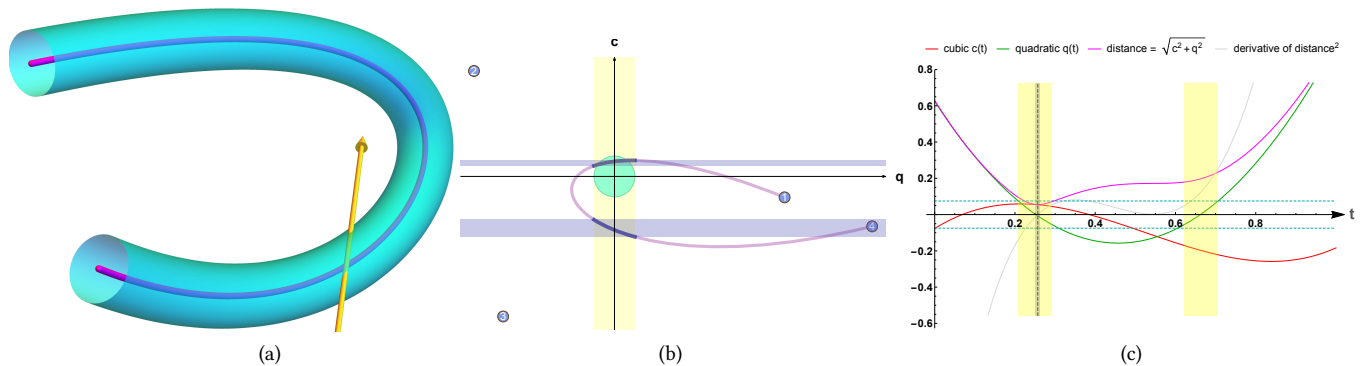


Figure 1: We restate a 3D ray–cubic Bézier curve intersection (a) as a 2D problem (b). The ray can only intersect the thick curve if the cyan disk around the 2D coordinate system origin overlaps two rectangles obtained by intersecting the yellow vertical slab with the two horizontal slabs. We only need to search for the minimum distance between the ray and the curve in the thin gray interval (c).

ABSTRACT

We present a new approach to finding ray–cubic Bézier curve intersections by leveraging recent achievements in polynomial studies. Compared with the state-of-the-art adaptive linearization, it increases performance by 5–50 times, while also improving the accuracy by 1000X. Our algorithm quickly eliminates parts of the curve for which the distance to the given ray is guaranteed to be bigger than a model-specific threshold (maximum curve’s half-width). We then reduce the interval with the isolated distance minimum even further and apply a single iteration of a non-linear root-finding technique (Ridders’ method).

CCS CONCEPTS

• Computing methodologies → Ray tracing; Parametric curve and surface models;

KEYWORDS

Bézier curves, ray tracing, hair and fur rendering, polynomial roots, Budan-Fourier theorem, Vincent’s theorem, VCA, VAG, VAS

ACM Reference format:

Alexander Reshetov. 2017. Exploiting Budan-Fourier and Vincent’s Theorems for Ray Tracing 3D Bézier Curves. In *Proceedings of HPG ’17, Los Angeles, CA, USA, July 28–30, 2017*, 11 pages. DOI: 10.1145/3105762.3105783

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HPG ’17, Los Angeles, CA, USA

© 2017 ACM. 978-1-4503-5101-0/17/07...\$15.00
DOI: 10.1145/3105762.3105783

1 INTRODUCTION AND PRIOR ART

The extraordinary success of computer graphics has largely been built on linear representations of real-world objects. This has allowed full hardware support, but it is on the wrong side of the *memory-vs-computation* trade-off. Fundamentally, to improve approximation requires finer and finer tessellation, resulting in significant memory utilization but diminishing quality improvements.

Perhaps nowhere is this predicament more noticeable than in hair and fur rendering. Using a mesh representation for such models is prohibitively expensive. Thus less-accurate texture approximations have typically been used in practice [Hadap et al. 2007; Kajiya and Kay 1989; Lengyel et al. 2001; Ren et al. 2010; Sintorn and Assarsson 2009]. This is changing now, driven by the adaptation of the content creation tools that can design genuine higher order surfaces and curves. The logic of asset management and production development favors ray tracing algorithms that allow more streamlined rendering pipeline [Christensen and Jarosz 2016]. Directly rendering hair and fur in ray tracing systems also benefits from the efficient occlusion culling due to customary front-to-back processing order. On the flip side, handling different levels-of-detail [Kang et al. 2013] is less compelling in a ray-tracing context as it affects acceleration structures [Djeu et al. 2011]. Solutions without LODs are more fitting, provided there is a time budget for exact intersections.

A separate—but very important—area of curve rendering research pertains to the 2D domain, with applications in text, decals, vector graphics, and texture rendering. It is anchored by the seminal work of Loop and Blinn [2005] and thoroughly explored by other researchers [Batra et al. 2015; Ganacim et al. 2014; Kilgard and Bolz 2012; Liao et al. 2012; Nehab and Hoppe 2008, 2012; Qin et al. 2008; Ray et al. 2005; Reshetov and Luebke 2016; Sen 2004; Sun et al. 2012; Tarini and Cignoni 2005]. Such methods are not directly transferable to 3D, and generally aim at evaluating a distance from

a given point sample to a curve—but not the parameter of the curve that corresponds to the minimum distance.

Sederberg and Nishita proposed to use convex hull property for Bézier clipping in 2D case [1990]. Their method is rather general, allowing to find intersections of two arbitrary coplanar curves.

The state-of-the-art in 3D geometric hair modeling was established by Nakamaru and Ohno [2002] and extended by Barringer et al [2012], Qin et al [2014], Woop et al [2014], and Chiang et al [2015]. In these approaches a parametric curve space is dynamically split until a curve fragment can be safely approximated with a straight line under given circumstances.

Consequently, such methods are faster for slowly changing or deeply subdivided curves. In contrast, our approach exhibits the opposite behavior: it is most potent for spatially varying cubic curves and becomes less efficient for almost linear curves (requiring an additional logic for a strictly quadratic or linear ones). And it is at its worst for axis-aligned straight lines. We demonstrate this factual *modus operandi* by comparing our approach with the adaptive linearization using version 3 of the *Physically Based Rendering* system by Pharr et al. [2016] for experiments. For free-form randomized Bézier curves, we improve the performance of the PBRT ray-curve intersection kernel by 50X; for all the models in the book the range is 6X–12X; while for an artificial model, consisting entirely of axis-aligned straight curves, our advantage is only 2.5X. Straight lines could easily be assigned to a separate branch of the algorithm, but we assume that it is the application’s responsibility to handle such cases (especially when hardware-accelerated facilities are readily available).

Other than the straight-line case, the reasonable performance and accuracy of our algorithm makes it suitable for a variety of applications such as hair, fur, cloth, or grass rendering. Our implementation can be used as a drop-in replacement, but is modular in nature, enabling its use in parts to accelerate other tasks. In particular, since we quickly eliminate parametric intervals on which the ray is guaranteed to be further from the curve than a given distance, our technique can jump-start more evolved approaches with assiduous shape modeling [Bronsvort and Klok 1985]. We also completely circumvent the LOD handling problem, computing exact intersection at any distance. This lets us use the same geometry regardless of the sampled path.

Determining a ray–curve proximity (*aka* intersection) is just a first step that feeds into a full rendering system. It requires a preprocessing step (building an acceleration structure) and post-processing step (actual shading). Given the wide-ranging nature of the related problems, we will limit our scope to only the ray–curve intersection kernel itself, while noting the excellent work of other researchers in this area [Andersen et al. 2016; Chiang et al. 2015; d’Eon et al. 2011; Moon et al. 2008; Ou et al. 2012; Qin et al. 2014; Woop et al. 2014; Wu and Yuksel 2017; Yan et al. 2015; Yu et al. 2012; Zinke et al. 2008].

It is expedient to consider parametric representation of rays and curves, which permits reformulating the related geometric problems as a solution of polynomial equations for such parameters. The original *Graphics Gems* [Glassner 1990] contains many excellent articles on the subject. In particular, Hook and McAree [1990] introduced to graphics community Sturm sequences that can be used

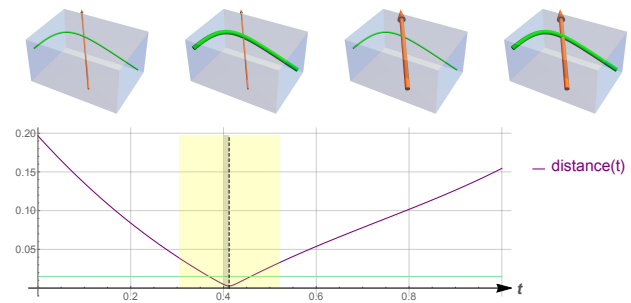


Figure 2: Case study for ray-curve intersections. Top: both ray and curve could have thickness. Bottom: distance between the ray and points on the curve as a function of the curve’s parameter t . The green line corresponds to the curve’s half-thickness.

to bracket real roots of univariate polynomial equations. Sturm’s theorem was, indeed, the method of choice for numerically solving polynomial equations for most of the 20th century as it was considered superior to earlier approaches based on Vincent’s theorem. Ironically, there was a complete turnaround later in the century, facilitated by the discovery of the efficient ways to compute Vincent sequences, the fastest one being the Vincent–Akritas–Strzeboński algorithm [2008].

2 RAY-CURVE PROXIMITY QUERIES

For an arbitrary curve and a ray—considered as objects with zero width—a probability of their exact intersection is 0. Depending on a specific application’s requirements, ray, curve, or both may have a thickness (Figure 2). However, it is more conventional and straightforward to search for intersections of zero-width rays with primitives whose cross-sectional geometry is defined by the model. In particular, cylindrical or elliptical profiles might work for a strand of hair, and ribbons for a blade of grass. Assuming a significant disparity between longitudinal and cross-sectional dimensions of such objects, a probability of their intersection with the ray that penetrates their bounding box will still be rather low (see section 2.2 for the study).

Therefore it is important to eliminate non-intersecting cases as quickly as possible. It can be facilitated by using squared distance between points on a curve and a ray (which is a sextic polynomial for a cubic Bézier curve). By analyzing properties of such function, we can efficiently find intervals on which the distance is guaranteed to be greater than the maximum of curve’s half-width (areas outside yellow intervals in Figure 2). This assertion does not even require computing the polynomial coefficients in an explicit form, let alone finding its roots.

Such interval reduction attends to all potential proximity queries, whether we are interested in finding bona fide ray–thick curve intersections or just want a distance minimum. In later cases, we able to reduce the interval even further (to the thin gray area). This lends itself well to the final step of our algorithm—actually finding a minimum through a single iteration of a local minimization technique (see section 2.8 for the comparison and evaluation of the possible approaches).

We have found that this yields the minimum position at $t = 0.412$. Note that the ray would intersect the swept cylinder around the curve (which radius is shown as green line) at two positions that are close but different from this one. One of these positions is for the ray entering the cylinder, another—exiting. As it may happen, an application might only be interested in the coordinate of the minimum, parameterized by curve’s t -value, as is the case of the PBRT shading system. We surmise that it is also possible to reduce intervals for ray/cylinder intersections exploiting sextic properties, similar to what we did for a minimum distance, which requires computing a quintic root. However here we concentrate our efforts on an efficient computation of the distance minimum that can be directly fed into existing shading systems.

2.1 Algorithm and Paper Outline

We are interested in finding minima of a distance between a curve and a ray that are smaller than a given threshold (which can be a function of curve’s parameter). We will casually refer to this as “intersection”. In most cases, such intersection will not exist, allowing a quick resolution of the query.

The first four steps of our algorithm aim at reducing the interval that can potentially contain the intersection, the last one at actually finding one. We proceed as follows (indexing the list by the corresponding paper section):

- 2.3 Though intrinsically three-dimensional, finding minimum distance(s) between a given ray and a curve can be restated as a 2D problem. In this new dual space, we minimize distance between points on the transformed Bézier curve and the coordinate system origin. As an added bonus, the x -coordinate of the new Bézier curve is a quadratic polynomial (and y is a cubic).
- 2.5 By exploiting properties of such polynomials (section 2.4), we eliminate the intervals on which the distance is guaranteed to be bigger than a given threshold (calculated from the curve’s cross-sectional profile).
- 2.6 Eliminate the cases for which we know that parameter s , which gives the closest point along the ray $\mathbf{o} + s \mathbf{d}$, will be negative—if we actually calculate it, which we do not yet need to do.
- 2.7 Analyze the remaining intervals and estimate the number of the extrema of the distance function (roots of its derivative). Intervals with more than one extremum are split further. Intervals with a single maximum are discarded. For the scenes we tested, there will only be a single minimum in 99.99% cases. For such occurrences we will reduce the interval even more—without using any root finding algorithm. Such reduced intervals are shown with gray shading in Figures 1c and 2.
- 2.8 Find a root after a single iteration of a local minimization technique applied to the reduced interval. Among such methods, those that employ a non-linear transformation of the target function allow a better accuracy.

In our experiments, when a curve’s radius is set to 1% of the maximum side of its bounding box, we eliminate 95% of all intersections just by analyzing polynomial coefficients. In the remaining 5% cases, the distance is less than the radius in 80% of them (i.e. intersection

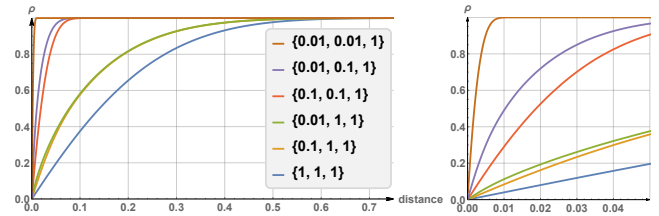


Figure 3: A probability of a random ray to be closer to a random curve than a given distance. Six different cases are studied, delineated by the size of the curve’s bounding box. We consider only the rays that intersect the curve’s bounding box and originate outside it.

cannot be eliminated, see section 2.2 for the detailed analysis). For the tested models in Table 1, we achieve this elimination in 93–99% cases.

Steps 2.3–2.5 of our algorithm have a geometric interpretation in which the ray can be separated from parts of the curve by axis-aligned lines in the dual space (ones that delineate the shaded slabs in Figure 1b). Other steps are algebraic in nature and do not allow such a simple geometric formulation. In the current implementation, these steps are only applicable to the cubic curves and are less general than the convex hull clipping.

2.2 Statistical Distance

For an arbitrary triangle, we have computed that a ray has a 28% probability to intersect a triangle if its origin is located inside the triangle’s bounding box, and 14% otherwise (appendix B.7). For a typical ray–curve intersection query, percentages are much lower.

We find a probability distribution by constructing a random Bézier curve with control points in a unit cube. We then compute a tight bounding box for the curve (not for its control points), create a random ray that intersects this box, and find a minimum distance between the ray and the curve. To normalize results, we also scale everything up so the largest side of the bounding box is 1.0.

The cumulative distribution function for such measurements is given by the blue curve in Figure 3. That is, a ray will be closer to a curve than 0.02 with 8% probability, closer than 0.01 with 4% probability, and so on. This blue graph accounts for a general case. Statistics will be different for flatter curves. We split all data, separately accumulating statistics for bounding boxes not exceeding given sizes. For example, when the two smaller sides of the box are less than 0.1 in length, distance between a ray and a curve will be less than 0.01 in 24% cases (red curve). And, of course, if two box sides are less than 0.01, the maximum distance cannot exceed $0.01\sqrt{2}$ (brown graph).

For rays with an origin inside a curve’s bounding box (which is typical for secondary rays), these probabilities are cut in half. We provide measurements for real scenes in section 3. These numbers highlight the importance of having an efficient culling strategy for the non-intersecting cases.

2.3 Dimensionality Reduction

A point on a cubic Bézier curve is given by the univariate polynomial $\mathbf{b}(t) = \mathbf{u}_0 + \mathbf{u}_1 t + \mathbf{u}_2 t^2 + \mathbf{u}_3 t^3$, where bold variables are

3D vectors and $t \in [0, 1]$ is a scalar parameter. This can be easily converted to and from the Bernstein form (9). Using matrix notation:

$$\mathbf{b}(t) = \mathcal{U} \mathbf{t}^T = [\mathbf{u}_0 \ \mathbf{u}_1 \ \mathbf{u}_2 \ \mathbf{u}_3] [1 \ t \ t^2 \ t^3]^T \quad (1)$$

We will use a vector of t -powers \mathbf{t} , 4×3 matrix of coefficients \mathcal{U} , as well as a modified \mathcal{V} (defined in 3) in other parts of the algorithm.

A distance from $\mathbf{b}(t)$ to a line that contains \mathbf{o} and has a unit direction \mathbf{d} is the length of the cross product vector $\mathbf{b}_2(t) = (\mathbf{b}(t) - \mathbf{o}) \times \mathbf{d}$ (*cathetus* = *hypotenuse* $\sin \alpha$). Each component of the 3D vector $\mathbf{b}_2(t)$ is a cubic polynomial and thus defines another Bézier curve. This new curve is shown in Figure 1b together with its control points. By design, vector coefficients of $\mathbf{b}_2(t)$ are orthogonal to \mathbf{d} and lay in the plane that contains origin $[0 \ 0 \ 0]$. We refer to this as “dual space” to the original 3D Euclidean space. We define two orthonormal vectors in this space as

$$\begin{aligned} \mathbf{q} &= \frac{\mathbf{u}_3 \times \mathbf{d}}{|\mathbf{u}_3 \times \mathbf{d}|} \\ \mathbf{c} &= \mathbf{q} \times \mathbf{d} \end{aligned} \quad (2)$$

These two vectors establish a coordinate system in the plane (shown as \mathbf{q} and \mathbf{c} axes in Figure 1b). If the vector $\mathbf{u}_3 \times \mathbf{d}$ has zero length, we choose an arbitrary unit vector in the plane for \mathbf{q} . We then compute

$$\begin{aligned} \mathcal{V} &= [\mathbf{u}_0 - \mathbf{o} \ \mathbf{u}_1 \ \mathbf{u}_2 \ \mathbf{u}_3] \\ \mathbf{p}_q &= \mathbf{q} \mathcal{V} \\ \mathbf{p}_c &= \mathbf{c} \mathcal{V} \end{aligned} \quad (3)$$

In the coordinate system (\mathbf{q}, \mathbf{c}) , four 2D vector coefficients for $\mathbf{b}_2(t)$ could be found by transposing 2×4 matrix $[\mathbf{p}_q \ \mathbf{p}_c]$ —but we have found more practical to work with 4D vectors \mathbf{p}_q and \mathbf{p}_c directly (5). Such approach yields operations well-suited for modern architectures.

Figure 4 shows the original 3D and the transformed 2D curve together using the same data as in Figure 1, as well as \mathbf{q} and \mathbf{c} axes.

By design, vectors \mathbf{q} , \mathbf{c} and \mathbf{d} establish an orthonormal coordinate system in 3D Euclidean space. Adaptive linearization algorithms also use the coordinate system with z -axis defined by the ray’s direction. By transforming everything into the ray-centric coordinate system, further calculations are simplified at the expense of some initial computations.

Our approach differs by specifically choosing \mathbf{q} axis in such a way that the cubic term in \mathbf{p}_q vanishes (as it is proportional to $(\mathbf{u}_3 \times \mathbf{d}) \cdot \mathbf{u}_3$, see also appendix B.6). In many cases, it allows meaningful interval reduction by itself (section 2.5), without computing other terms.

2.4 Polynomial Properties

This section gives a brief description of the mathematical apparatus used throughout the paper. The specific details are provided in appendices A and B. We address two related yet distinct problems for a polynomial $p(t)$ and its derivative $p'(t)$ on the interval $[t_1, t_2]$ using threshold r :

1. Find number of $p'(t)$ roots on $[t_1, t_2]$.
2. Verify that $p(t) < -r$ or $p(t) > r$ for all $t \in [t_1, t_2]$.

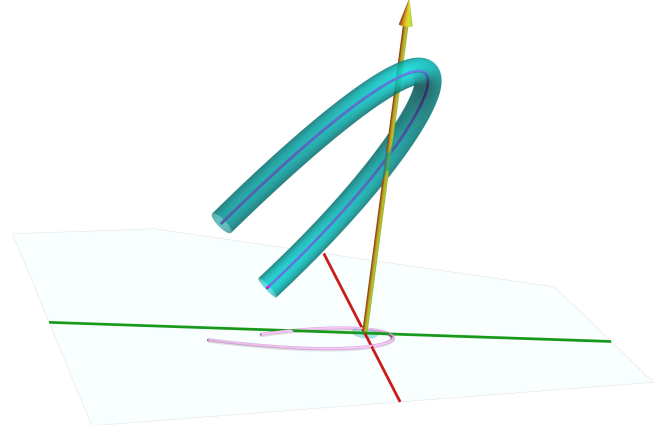


Figure 4: 3D curve and dual 2D curve shown together. The distance from a point on the 3D curve to the ray is equal to the distance from the corresponding point on the 2D curve to the coordinate origin $[0 \ 0]$ at the intersection of green (quadratic terms) and red (cubic) axes.

These tests are performed by analyzing numerical sequences derived from Budan-Fourier theorem and Vincent’s theorem. The former requires computing all derivatives of the given polynomial at t_1 and t_2 . For each such sequence, we count the number of sign variations. The difference between these two values yields the upper bound on the number of roots of the polynomial on the interval [Bensimhoun 2013].

The modern rendition of Vincent’s theorem provides an alternative—and more accurate—way of evaluating such a bound by computing the number of sign variations in the coefficients of the transformed polynomial

$$(1+x)^n p\left(\frac{t_1+t_2x}{1+x}\right)$$

where n is the degree of the analyzed polynomial $p(t)$.

In different parts of the paper, $p(t)$ is either a squared length of a 2D vector, or one of the components of such vector (section 2.5), or a ray parameter for the minimum (s in $\mathbf{o} + s \mathbf{d}$, see section 2.6).

If there is a single solution of $p'(t) = 0$ and the second test in (4) fails, i.e. we cannot guarantee that $\min_{t \in [t_1, t_2]} |p(t)| > r$, we find the root with any suitable technique (2.8). It will give us the sole minimum of $p(t)$ on the interval (maximum can be quickly dismissed by requiring $p'(t_1) < 0$).

If test 1 returns 0 roots, we immediately reject the interval even if $|p(t_1)| < r$ or $|p(t_2)| < r$. This simply means that the minimum is achieved at the adjacent intervals which will be handled in a due course. This is true for a single curve and for C^1 smoothly joined multiple curves as well (which is usually the case in hair modeling). If the curves are not connected smoothly—and this is an application specific case—we additionally check $p(0)$ or $p(1)$ if $t_1 = 0$ or $t_2 = 1$.

The calculations spent on test 1 do not contribute directly to the main goal of finding the intersection. Moreover, it affects the program flow only if more than one root is predicted. We have found that this is a rare event in practice (0.01% in the tested models). It would be frugal to reuse some of this data and it is indeed possible.

Figure 5 shows sextic polynomial $p(t)$ and its derivative. This is, in fact, the squared distance function for the left yellow interval in

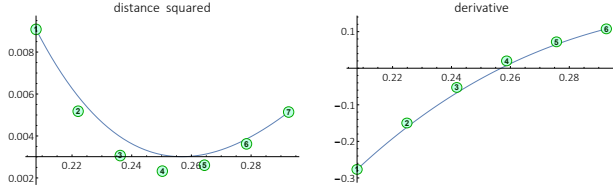


Figure 5: Vincent sequences for sextic $p(t)$ and its derivative. We use such sequences to make sure that there is only a single minimum on the interval and bracket it more precisely.

Figure 1c (one that was not excluded by other means). The minimum of this function corresponds to the closest distance between a curve and a ray.

First, we compute the Vincent sequence for quintic $p'(t)$ (right) on the given interval. The first entry in the sequence is $p'(t_1)$, the last one is $p'(t_2)$. The four middle entries do not correspond to any t , we just plot them equidistantly for convenience. Vincent's theorem states that $p'(t)$ has a single root if and only if the corresponding sequence has one sign variation. This is, indeed, our case: the sequence changes signs between third and fourth entries.

Second, we compute the Vincent sequence for $p(t)$ (left). This can be done cheaply utilizing partial sums of the sequence for $p'(t)$. The minimum of p -sequence (fourth entry) gives us a lower bound of $p(t)$ values on the processed interval. That is, the distance between the ray and the curve is greater than $\sqrt{0.0023} = 0.048$. If the modeled hair half-width were smaller than this value, we would have rejected the intersection (for a better visibility we use $r = 0.075$ in the examples, which is a rather thick hair).

Finally, the position of the p -sequence minimum gives us a good guess of the true minimum as $t_{new1} = t_1 + (t_2 - t_1)(4 - 1)/6$. Equipped with the fact that $p'(t)$ changes sign only once on the interval, we can bracket the true minimum by computing $t_{new2} = t_{new1} + (t_2 - t_1)/12$. If the signs of $p'(t_{new1})$ and $p'(t_{new2})$ differ, then the interval $[t_{new1}, t_{new2}]$ is a smaller bracket for the root, otherwise we check a bigger offset from t_{new1} .

The same logic is applied to the processing of the cubic component of the dual curve (section 2.5) and the disposition of the negative ray directions (section 2.6).

2.5 Interval Reduction

Distance between a point on the curve $\mathbf{b}(t)$ (1) and the ray $\mathbf{o} + s\mathbf{d}$ is equal to the length of the 2D vector $\mathbf{g}(t)$ with cubic coefficients (3):

$$\begin{aligned} \mathbf{t} &= [1 \ t \ t^2 \ t^3] \\ \mathbf{g}(t) &= \mathbf{t} [\mathbf{p}_q \ \mathbf{p}_c] \end{aligned} \quad (5)$$

Let r be the maximum radius of the curve's cross-sectional profile ("half-width"). We can guarantee that $|\mathbf{g}(t)| > r$ if $|g_x| > r$ or $|g_y| > r$. The squared length of the vector \mathbf{g} is a 6th degree polynomial of t . Instead of directly solving the roots of its derivative, we try to quickly refute the intersection by looking on the individual components of \mathbf{g} —without explicitly computing the sextic coefficients.

Since the x -component of the vector \mathbf{g} is a quadratic polynomial of t , we can easily find (one or two) intervals of t for which $|g_x| \leq r$. In Figure 1c, g_x is shown as the function $q(t)$. Its intersections with

$\pm r$ dotted cyan lines yield two yellow t -intervals that can potentially contain the minimum distance smaller than r . We have found that in about 30% of cases, such intervals will not exist at all or will be outside $t \in [0, 1]$ and we immediately reject the intersection.

For each unresolved interval $[t_1, t_2]$ we recompute r in hopes of reducing it, and then look onto g_y . It is a cubic function and its extrema could be obtained by solving a quadratic equation, allowing us to find $\min_{t \in [t_1, t_2]} |g_y(t)|$ exactly.

In practice we use an even simpler approach (A.3). Four components of the Vincent sequence \mathbb{C} for the polynomial $g_y(t)$ can be found by multiplying \mathbf{p}_c to the matrix of t_{12} -values \mathcal{M} given by (7). Intervals for which $\mathbb{C} > r$ or $\mathbb{C} < -r$ can be safely rejected. These calculations require no branching. On top of it, we will reuse the matrix \mathcal{M} to eliminate the cases in which the minimum distance is achieved for the negative ray directions.

Note that our algorithm does not change if the half-width r is a quadratic function of t and we use it directly instead of $\max_{t \in [t_1, t_2]} r(t)$. We could not analyze such a case because of our lack of compelling models. Given that $r(t)$ is typically small with respect to the curve's length, it will not have a significant impact on performance anyway.

2.6 Disposition of Negative Ray Directions

In most applications only positive values of the parameter s along the ray $\mathbf{o} + s\mathbf{d}$ are taken into account. Thus we can immediately dismiss the interval $[t_1, t_2]$ for which we know that $s < 0$ for all the closest points along the ray. This elimination is more likely for secondary rays.

A value of s for the point on the ray closest to $\mathbf{b}(t)$ can be computed as $\mathbf{d} \cdot (\mathbf{b}(t) - \mathbf{o})$. It is a cubic polynomial of t with coefficients $\mathbf{d}\mathcal{V}$, where \mathcal{V} is given by (3). Next we multiply the resulting vector of cubic coefficients by the matrix \mathcal{M} (7). If all four components of the resulting vector are negative, s is guaranteed to be less than 0 for all $t \in [t_1, t_2]$ (A.3).

This is a very simple check and we always do it. If the ray's origin is outside the bounding box of the curve, this test is unlikely to succeed, but ray/box intersection data is not saved in our testbed platform PBRT. This is a conservative test that may not resolve all cases. Once the actual value of s for the intersection is found, it must be retested again.

2.7 Root Isolation

If all our early rejection tests fail, we arrive at the step of our algorithm where we finally need the sextic polynomial $p(t)$ which is the squared norm of the vector \mathbf{g} in (5). $\sqrt{p(t)}$ yields the distance between point $\mathbf{b}(t)$ on the curve (1) and the given ray. To find $p(t)$, we compute dot products of \mathbf{p}_q and \mathbf{p}_c with vector \mathbf{t} and then $g_x^2 + g_y^2$. The first derivative $p'(t)$ can also be more easily evaluated as $2(g_x'g_x + g_y'g_y)$, without using sextic coefficients as such.

To compute the Fourier (A.1) and Vincent (A.5) sequences that we use in tests (4), we need all six non-zero derivatives of $p(t)$. There are two ways to handle this: *a*) continue differentiating individual components of \mathbf{g} or *b*) explicitly store and use seven $p(t)$ coefficients.

As it turns out, both methods have performed similarly, taking advantage of the reduced computations for the higher derivatives and the fact that $g_x(t)$ is a quadratic polynomial. Indeed, $p^{(6)}(t)$ is

just $720 p_c[3]^2$ (indexing vector from 0). Since we only need the sign of the derivatives, we could cancel the numerical multipliers. In case of the 6th derivative, however, we do not need to compute it at all, since it is always positive.

In practice we use method (a) for $p(t)$ and $p'(t)$ and (b) for the higher derivatives. This allows us to compute and store only the last five sextic coefficients since the first two are needed only for $p(t)$ and $p'(t)$. Once we have machinery for evaluating derivatives, we:

1. Compute Fourier sequences (A.1) at t_1 and t_2 . If analysis of their sign differences tells us that there are no roots on the interval, we reject it.
2. Convert Fourier sequences at the endpoints to the Vincent sequence for the whole interval (A.4). We will also compute a new bound for the number of roots if more than one root was predicted by Fourier sequences (which are less precise in this regard). We also further split intervals with more than one root. (6)
3. Reduce and potentially reject the interval using the technique we described in section 2.4.
4. And, finally, find a sole minimum of $p(t)$ as a root of $p'(t)$ on the reduced interval as explained in the next section.

Once we have found the minimum, we compare it to r and check that the ray goes in the positive direction.

One important caveat, pertinent to any root isolation technique, is the handling of very close roots. In our case, such roots may result in excessive splits trying to separate them. From a pragmatic point of view, however, if roots are close enough to each other, they are all good (see example in right part of Figure 7). For this reason, we avoid splitting small intervals and proceed with the root finding part of the algorithm at once.

2.8 Root Finding

Various root-finding techniques exist: we refer readers to Wikipedia article from which we took our inspiration [2016]. Figure 6 provides maximum and average errors in distance computation after a single iteration for a variety of different approaches. We carried out multiple tests for random curves and rays in a unit cube and aggregated the results. The distance error is a more reliable yardstick, since the maximum error in curve's t -parameter can be unbounded (e.g. in case of multiple intersections or when a ray is parallel to a straight line curve). On the other hand, the average t -error behaves similarly to the average d -error.

Notably, the well-known Newton–Raphson method has the worst accuracy, followed by its discrete companion—secant method. These two methods would benefit from the almost linear behavior of the examined function, as they use x -intercept of the function's tangent line. This is not the case for our target function $f(t) = p'(t)$, which is a derivative of the squared distance $p(t) = g(t) \cdot g(t)$ in (5). Even if the original $b(t)$ in (1) is not varying wildly, the act of $b(t) \rightarrow g(t)$ projection could exacerbate the situation.

Under such circumstances, methods that expect non-linear target function have an edge. Muller [1956] finds roots of a parabola

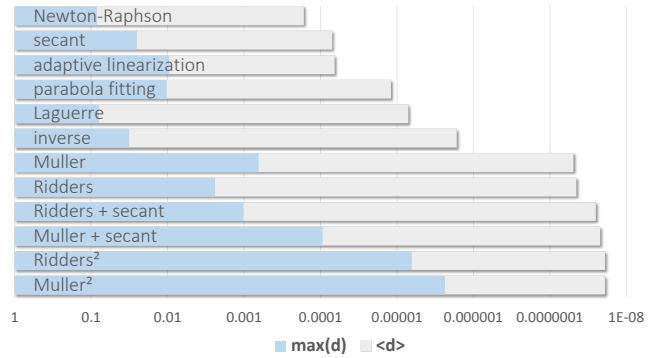


Figure 6: Maximum and average errors in distance computations for different minimization techniques (the longer the bar the smaller the error).

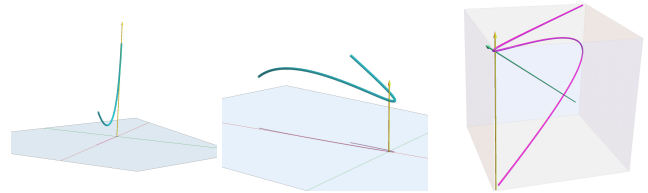


Figure 7: Worst cases for t (left) and distance (center) after 10^9 simulations. Right: Klassen's curve with control points $\{-1, -1, -1\}$, $\{5, 5, 1\}$, $\{-5, -5, 1\}$, $\{1, 1, 1\}$ and rays with the worst distance and t errors.

that interpolates $f(t)$ values at $t = \{t_1, (t_1 + t_2)/2, t_2\}$; Ridders [1979] probes an exponential approximation to linearize such values. These two methods are superior even in comparison with multiple iterations of the simpler techniques (Newton's and secant), also taking advantage of the already computed p' at the endpoints.

To pick one, Ridders' is more straightforward and requires no branching, while Muller's has to check if the parabola is degenerated into a line.

Ridders' method finds t_{new} that is guaranteed to be inside $[t_1, t_2]$. Among the four values $t_1, (t_1 + t_2)/2, t_2$, and t_{new} , we could always choose two consecutive ones (let's say t_a and t_b) with opposite p' signs. We can then promptly fine-tune the result with the basic variant of the secant method as $(f(t_b) t_a - f(t_a) t_b) / (f(t_b) - f(t_a))$. It reduces the worst error, while having almost no impact on the performance. We use this combination for all the reported results.

Figure 7 shows ray/curve data resulting in the worst error in t (2.7×10^{-2}) and distance (1.67×10^{-2}) that were found after 10^9 tests. Interestingly, we found that the worst-distance curve is similar to the one suggested by Klassen [1991], which causes the most problems in numerical methods.

Two iterations of Muller's or Ridders' methods (the last two bars in the chart) would significantly reduce errors even for such curves. For most applications though it is an over-kill.

Other tested methods—Laguerre's, parabola fitting for $p(t)$, and inverse quadratic interpolation—are neither simple nor particularly accurate for our problem.

Figure 6 also shows results for the adaptive linearization (using PBRT's implementation verbatim and executing the prescribed

number of iterations). This technique estimates the distance error to find the number of run-time splits for the curve. For random curves this approach sometimes falter since the error is estimated only once for the whole curve, but for real models it is mostly adequate as evident from Table 1.

The adaptive linearization error can be reduced by prepensely increasing the number of splits—with the detrimental impact on performance. Ironically, we cannot use less-accurate-but-faster root finding algorithm in our approach to get a sizable performance boost. Using the simplest secant technique improves the performance for the tested models by only 3%, while still having 8X better accuracy than the default setting for the adaptive linearization.

3 RESULTS

Table 1 compares performance and accuracy of our ray-curve intersection technique with PBRT, measured on a second generation Intel i7-3930K at 3.2GHz. Only the kernel measurements are given in the table, ignoring shading and acceleration structure traversal. The overall wall clock performance improvement is the smallest for the single hair model (1.3X) in which most of the time is spent traversing empty space. For other models it is in 2.3–5.4X range.

The measurements confirm contrarian characteristics of our algorithm which is less efficient for almost-linear curves. The performance gains shrink from 49.6X for the single curly hair model to 6.8X for the straight hair.

We did not change the default behavior of PBRT’s bounding volume hierarchy (BVH) builder, in which curves are automatically pre-split into 2^3 pieces, for models 2–6. For the single hair, which is comprised of 25 Bézier curves, we kept all these curves intact. This is the best setup to minimize the overall wall clock timing.

The disadvantage of the almost linear curve segments apropos of our algorithm is that for some rays the initial interval reduction will be rather paltry. For rays parallel to the prevalent segment direction (“hair splitting”), the total variation of the quadratic term g_x in (5) may be smaller than r , as exemplified in Figure 7 (left). The flip side of this is that it becomes possible to encapsulate the entire curve segment in a cylinder with a small radius. We implemented this functionality specifically for such cases. Even though for more spatially varying curves such checks are mostly extraneous, finding the distance between the ray direction and the cylinder axis is cheap.

Another component of our implementation that becomes essentially irrelevant for shorter segments is using exact curve bounds during BVH building. Each coordinate of a cubic Bézier curve could have only two extrema that can easily be found by solving a quadratic equation. This allows tight bounding box. Yet, for deeply split segments these bounds are just slightly smaller than ones for the control points of the split curve. We kept this functionality anyway as an instrument for future studies in BVH construction.

4 DISCUSSION

The ratio of longitudinal to cross-sectional sizes for a typical hair or fur filament is in 100s. This necessitates splitting such primitives to avoid a significant spatial overlap for dense models. We believe that such splits should be made dynamically, aligning split planes for nearby primitives, considering curves and triangles. This is

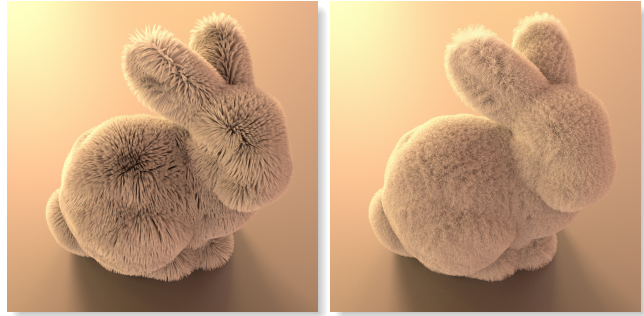


Figure 8: Two bunnies: the original PBRT model and one with the randomized fur.

a potential area for future research, also taking into account the dramatic changes in the cost function for the surface area heuristics that our algorithm brought in.

Orienting BVH nodes by a prevalent hair direction is also an attractive proposition [Woop et al. 2014], though its impact may vary depending on the model. We randomized each fur strand in PBRT’s furry bunny model (Figure 8). This produced a plush effect, simultaneously barring any potential improvements from the oriented bounding boxes.

Hair and fur is somewhere in between the traditional geometric objects, such as triangles, and purely volumetric constructs. This calls for further studies, in line with volumetric fur with explicit hair strands [Andersen et al. 2016] or distance aware ray tracing for curves [Nakamaru et al. 2012].

These are the most promising areas of research. We are also interested in accelerating intersections of rays and swept cylindrical shapes as well [Bronsvort and Klok 1985], using the new mathematical apparatus we proposed in this paper.

A ROOT LOCALIZATION TECHNIQUES


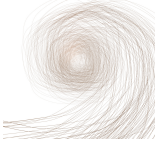
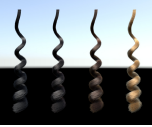



If a polynomial function has a single root on a given interval, it can be reliably found. For this reason, it is important to isolate the polynomial roots. All such techniques stem from the original Descartes’ rule of sign [1637].

There is a significant body of work on this subject, summarized by Bensimhoun [2013] and Biagioli [2016]. We are interested in the two particular formulations (4) for polynomials of degree 3, 5, and 6; their Vincent sequences will be referred as \mathbb{C} , \mathbb{Q} , and \mathbb{S} . With respect to finding (an upper bound of) number of roots for $p'(t) = 0$, the utile answers are 0, 1, and “more than one”. The second problem of proving that $|p(t)| > r$ can be facilitated by showing that two equations $p(t) = \pm r$ have no roots.

Methods based on Sturm’s theorem yield the exact number of roots yet require long polynomial division. Fourier sequences are easy to compute but can overshoot the actual number of roots. The modern techniques, such as Vincent–Collins–Akritas [1976], Vincent–Alesina–Galuzzi [2000], and Vincent–Akritas–Strzeboński [2008], are in the middle with respect to the efficiency and the simplicity.

Figure 9 shows the histogram distribution for the number of roots for arbitrary curves and rays on intervals that survived the tests in section 2.5. For real models, Fourier method reports 0 or 1

Table 1: Kernel performance and accuracy for the tested models. PBRT results with the default settings are given in gray. The last four columns show (a) percent of early exits (before minimum search), (b) percent of ray-curve intersections for rays that intersect curve's bounding box, (c) unresolved tests that do not have an intersection, and (d) average ratio of curve's half-width to its length for split segments. $a + b + c = 100\%$.

						
	1. single hair	2. lock	3. curls	4. straight	5. curly	6. fur
average kernel time in nanoseconds	13.6	33.1	57.0	116.0	145.6	90.8
average distance error	7.7×10^{-10} 2.3×10^{-6}	1.8×10^{-10} 6.8×10^{-7}	2.6×10^{-10} 9.8×10^{-7}	4.1×10^{-10} 7.4×10^{-6}	2.9×10^{-10} 4.5×10^{-7}	5.5×10^{-12} 5.9×10^{-7}
maximum distance error	2.7×10^{-8} 6.9×10^{-5}	8.3×10^{-9} 4.2×10^{-5}	1.2×10^{-5} 2.1×10^{-4}	9.7×10^{-7} 9.1×10^{-4}	1.7×10^{-5} 1.7×10^{-3}	2.0×10^{-8} 1.9×10^{-5}
a. culled tests	98.8%	98.2%	99.2%	93.4%	94.8%	96.6%
b. intersections	1.08%	1.60%	0.64%	5.03%	3.15%	2.55%
c. no intersections	0.12%	0.20%	0.16%	1.57%	2.05%	0.85%
d. average r /length	0.0022	0.0038	0.0038	0.018	0.062	0.033

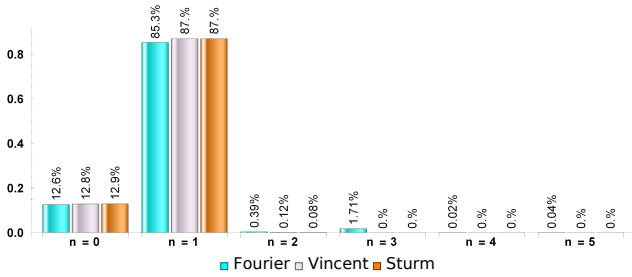


Figure 9: Number of roots reported by different techniques in randomized tests. Sturm's theorem always give the correct answer, while Fourier and Vincent techniques may overstate it by even number.

roots in 99.2% cases, and Vincent—in 99.99% cases. Therefore we start counting roots with the simplest Fourier method and switch to the Vincent one only in the case of the multiple reported roots.

We use *Mathematica* [2017] to derive formulæ in this appendix; the corresponding notebook is given in appendix B.

A.1 Budan-Fourier Theorem

Fourier sequence of a polynomial contains all its non-zero derivatives. To find extrema of 6^{th} -degree polynomial $p(t)$, we search for roots of its derivative $p'(t)$. Fourier sequence of $p'(t)$ is given by $\{p'(t), p''(t), p^{(3)}(t), p^{(4)}(t), p^{(5)}(t), p^{(6)}(t)\}$. We evaluate such sequences at both endpoints of the interval $[t_1, t_2]$ and then count the number of sign variations in each sequence. The difference between these two values constitutes the upper bound on the number of roots of $p'(t)$ on the interval [Bensimhou 2013].

Now let's consider a variable substitution $t = t_1 + x(t_2 - t_1)$ that transforms $p(t)$ defined for $t \in [t_1, t_2]$ into $h(x)$ defined for $x \in [0, 1]$. The chain differentiation rule allows computing Fourier sequences for $h'(0|1)$ as a product of $p'(t|t_2)$ sequences times

$\{dt, dt^2, dt^3, dt^4, dt^5, dt^6\}$ where $dt = t_2 - t_1$. This will come handy converting Fourier sequences to Vincent ones (A.4). We will also enjoy a certain latitude in scaling entries in Fourier sequences that do not change signs but simplify calculations.

A.2 Vincent's Theorem

We are mostly interested in a single application of Vincent's theorem. In rare cases when we do need to split the interval, we reset everything, compute a new Fourier sequence (at the split position) and then switch to Vincent's if warranted. In these circumstances, differences between particular implementations of Vincent's theorem (referred as VCA, VAG, or VAS) are not essential. VAG and VAS handle $[t_1, t_2]$ interval directly, while VCA requires a variable substitution as it deals with $t \in [0, 1]$.

A.3 Cubic Polynomial

For a cubic polynomial $c(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$ considered on interval $[t_1, t_2]$, we want to know if $|c(t)| > r$, i.e. equations $c(t) = \pm r$ have no roots on the interval. VAG method [Alesina and Galuzzi 2000] tells us to compute

$$(1+x)^3 \left(c \left(\frac{t_1 + t_2 x}{1+x} \right) - \rho \right)$$

where ρ is either $+r$ or $-r$. The coefficients of the resulting cubic polynomial of x define the Vincent sequence \mathbb{C} for $c(t) - \rho$ as

$$\mathcal{M} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ t_1 & (2t_1 + t_2)/3 & (t_1 + 2t_2)/3 & t_2 \\ t_1^2 & t_1(t_1 + 2t_2)/3 & t_2(2t_1 + t_2)/3 & t_2^2 \\ t_1^3 & t_1^2 t_2 & t_1 t_2^2 & t_2^3 \end{bmatrix} \quad (7)$$

$$\mathbb{C} = [c_0 \ c_1 \ c_2 \ c_3] \mathcal{M} - [\rho \ \rho \ \rho \ \rho]$$

The vector \mathbb{C} is obtained by multiplying the vector of cubic coefficients $c_{0123} = [c_0 \ c_1 \ c_2 \ c_3]$ by the matrix of t -values \mathcal{M} . We then

subtract ρ from each component. We also took a liberty of rescaling \mathbb{C} entries to simplify calculations, since we are only interested in signs. Vincent's theorem ensures that $|c(t)| > r$ if $c_{0123} \mathcal{M} > r$ or $c_{0123} \mathcal{M} < -r$ for all four components. Note also that the first and last components are polynomial values at the interval's endpoints. If $r = 0$, which is the case for the disposition of the negative ray directions (section 2.6), only one comparison is needed.

A.4 Fourier \rightarrow Vincent Conversion

Differentiation is linear with respect to polynomial coefficients. A polynomial and its Fourier or Vincent sequences have the same number of degrees of freedom. It is reasonable to assume that one representation can be converted to another. This is a correct assumption which can be verified by performing symbolic transformations. It is easier to do so when endpoints are 0 and 1.

Let a_i and b_i be Fourier sequences at $x = 0$ and $x = 1$ for quintic $h(x)$. Then its VCA sequence, which corresponds to the coefficients of a new polynomial $(1+x)^5 h(1/(1+x))$, can be found as either

$$\mathbb{Q} = \begin{pmatrix} a_1 + a_2 + \frac{a_3}{2} + \frac{a_4}{6} + \frac{a_5}{24} + \frac{a_6}{120} \\ 5a_1 + 4a_2 + \frac{3a_3}{2} + \frac{a_4}{3} + \frac{a_5}{24} \\ 10a_1 + 6a_2 + \frac{3a_3}{2} + \frac{a_4}{6} \\ 10a_1 + 4a_2 + \frac{a_3}{2} \\ 5a_1 + a_2 \\ a_1 \end{pmatrix} = \begin{pmatrix} b_1 \\ 5b_1 - b_2 \\ 10b_1 - 4b_2 + \frac{b_3}{2} \\ 10b_1 - 6b_2 + \frac{3b_3}{2} - \frac{b_4}{6} \\ 5b_1 - 4b_2 + \frac{3b_3}{2} - \frac{b_4}{3} + \frac{b_5}{24} \\ b_1 - b_2 + \frac{b_3}{2} - \frac{b_4}{6} + \frac{b_5}{24} - \frac{b_6}{120} \end{pmatrix} \quad (8)$$

Since we retain both Fourier sequences (A.1), we use only the shaded expressions to simplify computations. What remains is to find Fourier sequences for $h(x)$ at $x = 0$ and 1 using $p'(t)$ -sequences defined for $t = t_1$ and t_2 . We already know how to do it (multiply by powers of $dt = t_2 - t_1$). This allows us to analyze $p(t)$ and $p'(t)$ behavior on interval $[t_1, t_2]$ using the apparatus devised for Bézier curves on interval $[0, 1]$, which is described in the next section.

A.5 Sextic Polynomial and its Derivative

A cubic Bézier curve in Bernstein form is given by

$$\mathbf{b}(t) = (1-t)^3 \mathbf{w}_1 + 3(1-t)^2 t \mathbf{w}_2 + 3(1-t) t^2 \mathbf{w}_3 + t^3 \mathbf{w}_4 \quad (9)$$

Seven VCA coefficients for the sextic polynomial $p(t) = \mathbf{b}(t) \cdot \mathbf{b}(t)$ on $[0, 1]$ interval are computed using $\mathbf{w}_{ij} = \mathbf{w}_i \cdot \mathbf{w}_j$ as

$$\mathbb{S} = \begin{bmatrix} \mathbf{w}_{44} & \mathbf{w}_{34} & \frac{2\mathbf{w}_{24} + 3\mathbf{w}_{33}}{5} & \frac{\mathbf{w}_{14} + 9\mathbf{w}_{23}}{10} & \frac{2\mathbf{w}_{13} + 3\mathbf{w}_{22}}{5} & \mathbf{w}_{12} & \mathbf{w}_{11} \end{bmatrix} \quad (10)$$

We scaled down all the entries by the corresponding binomial coefficients $[1 \ 6 \ 15 \ 20 \ 15 \ 6 \ 1]$ to get expressions as linearly interpolated dot products of the control points \mathbf{w}_i . If each component of \mathbb{S} is greater than r^2 , then $p(t) > r^2$ for all $t \in [0, 1]$.

Sequential differences of \mathbb{S} are equal to the scaled VCA coefficients for quintic $p'(t) = 2\mathbf{b}'(t) \cdot \mathbf{b}(t)$ using the following identity

$$D(\mathbb{S}) = 1/6 \mathbb{Q} / [1 \ 5 \ 10 \ 10 \ 5 \ 1] \quad (11)$$

This gives a simple algorithm of computing \mathbb{S} when \mathbb{Q} is known, just by inverting the discrete differentiation (as partial sums of \mathbb{Q}).

Therein we shall emphasize that \mathbb{Q} is obtained from Fourier sequences for t_1 and t_2 but corresponds to $[0, 1]$ interval as proffered in the previous section. Thus the whole procedure has an effect of double splitting the original curve at t_1 and t_2 to produce the new Bézier curve defined on $[0, 1]$ that coincides with the original

curve for all $t \in [t_1, t_2]$. We could have done all this explicitly, but our approach is much simpler since we do not have to compute the control points \mathbf{w}_i of the new curve.

For practical purposes, we roll in all the numerical factors and also store the coefficients in the reverse order (from the highest degree to a free member), as illustrated in Figure 5.

B MATHEMATICA NOTEBOOK

This is a *Mathematica* [2017] notebook used to derive all formulae in this paper. For clarity, it is split into sections corresponding to appendix A. We define Bézier curves and polynomials as

```
(* cubic Bezier curve in Bernstein form as in equation 9 *)
bc3[t_, {p1_, p2_, p3_, p4_}] :=
  (1-t)^3 p1 + 3 (1-t)^2 t p2 + 3 (1-t) t^2 p3 + t^3 p4;
bc3'[t_, {p1_, p2_, p3_, p4_}] :=
  3 (1-t)^2 (p2-p1) + 6 (1-t) t (p3-p2) + 3 t^2 (p4-p3);
(* univariate polynomials and derivatives *)
poly[t_, c_List] := c[[1]]+Plus@@MapIndexed[#1+t^#2[[1]]&, Rest[c]];
poly'[t_, c_List] := D[poly[t, c], t];
```

B.3 Cubic Polynomial

```
c3 = Array[c, 4, 0]; (* coefficients *)
v3 = CoefficientList[Expand[(1+t)^3*(poly[(t1+t2 t)/(1+t], c3)-r)], t];
v3r = Expand[v3*{1, 1/3, 1/3, 1}] + r; (* doesn't depend on r *)
(* the same expression in matrix form *)
vm = Simplify@CoefficientList[v3r /. Thread[c3 -> u^Range[0, 3]], u];
vm = Transpose[vm];
Print@MatrixForm[vm /. t1 -> Subscript[t, 1] /. t2 -> Subscript[t, 2]]
Print@Simplify[c3.vvm - v3r] (* verify *)
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ t_1 & \frac{1}{3}(2t_1+t_2) & \frac{1}{3}(t_1+2t_2) & t_2 \\ t_1^2 & \frac{1}{3}t_1(t_1+2t_2) & \frac{1}{3}t_2(2t_1+t_2) & t_2^2 \\ t_1^3 & t_1^2 t_2 & t_1 t_2^2 & t_2^3 \\ \{0, 0, 0, 0\} \end{pmatrix}$$

B.4 Fourier \rightarrow Vincent Conversion

```
(* Bezier control points *)
bc = Table[Subscript[w, i], {i, 4}];
(* 10 dot products of bc terms using * for easier manipulations *)
bco = DeleteDuplicates@Flatten@Outer[Times, bc, bc];
(* bco as an array *)
bca = Table[Subscript[d, i], {i, Length[bco]}];
(* looking for roots of f[x] = g[x] in equation 5 *)
f[x_] := 2*bc3[x, bc]*bc3'[x, bc];
(* Vincent sequence for f (VCA form *)
vs = CoefficientList[Simplify[(1+t)^5 * f[1/(1+t)]], t]//Expand;
vsd = vs /. Thread[bco -> bca]; (* express vs using dot products *)
vsa = Table[Subscript[v, i], {i, Length[vs]}]; (* vsd as an array *)
(* Fourier sequence for f *)
fs = Simplify@Table[CoefficientList[D[f[t], {t, n}], t], {n, 0, 5}];

For[f2v = {}; tx = 0, tx <= 1, tx++, (* over {0,1} endpoints *)
  fsx = poly[tx, #]&/@fs; (* fs at endpoint *)
  fsd = Simplify[Expand[fsx] /. Thread[bco -> bca]]; (* as dot products *)
  fsa = Table[Subscript[a, b][[tx+1]], {i, Length[fsx]}]; (* array *)
  (* equations linking fsx and VCA terms *)
  eqn = Eliminate[Flatten[{Thread[fsd == fsa], Thread[vsd == vsa]}], bca];
  (* express VCA through Fourier sequence at tx *)
  AppendTo[f2v, Expand[vsa /. First@Solve[eqn, vsa]]];
];
(* choose the simplest expression (shaded terms) *)
ff = "Courier_New"; cyan = RGBColor[0.85, 1, 1];
Print@Row[
  Flatten[{Text[Style[#, FontFamily->ff]]&
    /@ f2v[[1, :, 3]],
    Text[Style[#, FontFamily->ff], Background->cyan]&
    /@ f2v[[1, 4, :]]}]]//MatrixForm,
```

```
"=",
Flatten[{Text[Style[#, FontFamily->ff], Background->cyan]&
/@ f2v[{{2, 3}],
Text[Style[#, FontFamily->ff]]&
/@ f2v[{{2, 4, 5}]}]//MatrixForm
}];
```

$$\begin{pmatrix} a_1 + a_2 + \frac{a_3}{2} + \frac{a_4}{6} + \frac{a_5}{24} + \frac{a_6}{120} \\ 5a_1 + 4a_2 + \frac{3a_3}{2} + \frac{a_4}{3} + \frac{a_5}{24} \\ 10a_1 + 6a_2 + \frac{3a_3}{2} + \frac{a_4}{6} \\ 10a_1 + 4a_2 + \frac{3a_3}{2} + \frac{a_4}{6} \\ 5a_1 + a_2 \\ a_1 \end{pmatrix} = \begin{pmatrix} b_1 \\ 5b_1 - b_2 \\ 10b_1 - 4b_2 + \frac{b_3}{2} \\ 10b_1 - 6b_2 + \frac{3b_3}{2} - \frac{b_4}{6} \\ 5b_1 - 4b_2 + \frac{3b_3}{2} - \frac{b_4}{3} + \frac{b_5}{24} \\ b_1 - b_2 + \frac{b_3}{2} - \frac{b_4}{6} + \frac{b_5}{24} - \frac{b_6}{120} \end{pmatrix}$$

B.5 Sextic Polynomial and its Derivative

```
Print@Text[bc3[t, bc]]
vp = CoefficientList[Simplify[(1+t)^6 (bc3[1/(1+t), bc]^2 - r^2)], t];
vp = Expand[vp/{1, 6, 15, 20, 15, 6, 1} + r^2]; (* doesn't depend on r *)
Print@MatrixForm[{{vp}/. Thread[Flatten@Outer[Times, bc, bc] ->
Flatten@Outer[Dot, bc, bc]]];
Print@Expand[Differences[vp] + 1/6 vs/{1, 5, 10, 10, 5, 1}];
```

$$\begin{pmatrix} (1-t)^3 w_1 + 3(1-t)^2 t w_2 + 3(1-t) t^2 w_3 + t^3 w_4 \\ (w_4 - w_4 w_3 - w_4 w_4 - \frac{2w_2 w_4}{5} + \frac{3w_2 w_3}{10} + \frac{9w_2 w_3}{10} - \frac{2w_1 w_3}{5} + \frac{3w_2 w_2}{5} w_1 - w_2 w_1 w_1) \\ \{0, 0, 0, 0, 0, 0\} \end{pmatrix}$$

B.6 p_q is a Quadratic Polynomial

```
ro = {ox, oy, oz}; rd = {dx, dy, dz}; (* ray's origin and direction *)
U = Array[u, {4, 3}]; (* matrix U for b(t) in equation 1 *)
q = Cross[U[[4]], rd]; (* q and c = q x rd from equation 2 *)
(* cubic term in p_q vanishes, as the next expression is 0 *)
Simplify@Last@CoefficientList[{{(1, t, t^2, t^3).U - ro}.q, t}
0
```

B.7 Probability of Ray-Triangle Intersection

```
r := RandomReal[{0, 1}, 3];
For[n = 1000000; nx = 0; i = 1, i <= n, i++,
{v0, v1, v2} = {r, r, r}; (* vertices *)
bb = MinMax /@ ({v0, v1, v2} // Transpose) // Transpose;
ro = bb[[1]] + (bb[[2]] - bb[[1]]) * r;
rd = Normalize[r - r];
nx += Boole[Abs@Total[Sign[#{[1]] - ro}.Cross[#[[2]] - ro, rd]] &
/ @ {{v0, v1}, {v1, v2}, {v2, v0}}] == 3];
];
Print["\[Rho]_=" , nx, "/", n, n];
```

ACKNOWLEDGMENTS

We are deeply grateful to Cem Yuksel and Benedikt Bitterli for providing a hair geometry which is used under a creative commons attribution license.

The availability of PBRT source code and the excellent documentation was instrumental in our research; we would like to congratulate Matt Pharr, Greg Humphreys, and Wenzel Jakob on a job well done.

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions.

REFERENCES

- Alkiviadis G Akritas, Andreas I Argyris, and Adam W Strzeboński. 2008. FLQ, the Fastest Quadratic Complexity Bound on the Values of Positive Roots of Polynomials. *Serdica Journal of Computing* 2, 2 (2008), 145–162.
- Alberto Claudio Alesina and Massimo Galuzzi. 2000. Vincent's Theorem from a Modern Point of View. *Rendiconti del Circolo Matematico di Palermo Serie II* (2000), 179–191.

- Tobias Grønbeck Andersen, Viggo Falster, Jeppe Revall Frisvad, and Niels Jørgen Christensen. 2016. Hybrid Fur Rendering: Combining Volumetric Fur with Explicit Hair Strands. *Vis. Comput.* 32, 6-8 (June 2016), 739–749.
- Rasmus Barringer, Carl Johan Gribel, and Tomas Akenine-Möller. 2012. High-quality Curve Rendering using Line Sampled Visibility. *ACM Trans. Graph.* 31, 6, Article 162 (Nov. 2012), 10 pages.
- Vineet Batra, Mark J. Kilgard, Harish Kumar, and Tristan Lorach. 2015. Accelerating Vector Graphics Rendering using the Graphics Hardware Pipeline. *ACM Trans. Graph.* 34, 4, Article 146 (July 2015), 15 pages.
- Michael Bensimhou. 2013. Historical Account and Ultra-simple Proofs of Descartes's Rule of Signs, De Gua, Fourier, and Budan's Rule. *ArXiv e-prints* (Sept. 2013). arXiv:math.HO/1309.6664
- Eric Javier Biagioli. 2016. *Methods for Bounding and Isolating the Real Roots of Univariate Polynomials*. Ph.D. Dissertation. Instituto Nacional de Matemática Pura e Aplicada.
- Willem F. Bronsvort and Fopke Klok. 1985. Ray Tracing Generalized Cylinders. *ACM Trans. Graph.* 4, 4 (Oct. 1985), 291–303.
- Matt Jen-Yuan Chiang, Benedikt Bitterli, Chuck Tappan, and Brent Burley. 2015. A Practical and Controllable Hair and Fur Model for Production Path Tracing. In *SIGGRAPH Talks*. ACM.
- Per H. Christensen and Wojciech Jarosz. 2016. The Path to Path-Traced Movies. *Foundations and Trends® in Computer Graphics and Vision* 10, 2 (2016).
- George E. Collins and Alkiviadis G. Akritas. 1976. Polynomial Real Root Isolation Using Descartes's Rule of Signs. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '76)*. 272–275.
- Eugene d'Eon, Guillaume Francois, Martin Hill, Joe Letteri, and Jean-Marie Aubry. 2011. An Energy-conserving Hair Reflectance Model. In *Proceedings of the Twenty-second Eurographics Conference on Rendering (EGSR '11)*. 1181–1187.
- René Descartes. 1637. *La géométrie (Discours de la Méthode, third part)*. Ed. of Leyde.
- Peter Djeu, Warren Hunt, Rui Wang, Ikrima Elhassan, Gordon Stoll, and William R. Mark. 2011. Razor: an Architecture for Dynamic Multiresolution Ray Tracing. *ACM Trans. Graph.* 30, 5, Article 115 (Oct. 2011), 26 pages.
- Francisco Ganacim, Rodolfo S. Lima, Luiz Henrique de Figueiredo, and Diego Nehab. 2014. Massively-parallel Vector Graphics. *ACM Trans. Graph.* 33, 6, Article 229 (Nov. 2014), 14 pages.
- Andrew S. Glassner. 1990. *Graphics Gems*. Academic Press, Inc., Orlando, FL, USA.
- Sunil Hadap, Marie-Paule Cani, Ming Lin, Tae-Yong Kim, Florence Bertails, Steve Marschner, Kelly Ward, and Zoran Kačić-Alesić. 2007. Strands and Hair: Modeling, Animation, and Rendering. In *ACM SIGGRAPH 2007 Courses (SIGGRAPH '07)*. 1–150.
- D. G. Hook and P. R. McAree. 1990. Graphics Gems. Chapter Using Sturm Sequences to Bracket Real Roots of Polynomial Equations, 416–422.
- Wolfram Research, Inc. 2017. *Mathematica*, Version 11.1. (2017). Champaign, IL, 2017.
- J. T. Kajiya and T. L. Kay. 1989. Rendering Fur with Three Dimensional Textures. *SIGGRAPH Comput. Graph.* 23, 3 (July 1989), 271–280.
- L. Kang, J. Wuxia, G. Guohua, and H. Yi. 2013. Level-of-Detail Modeling with Artist-Defined Constraints for Photorealistic Hair Rendering. In *2013 International Conference on Virtual Reality and Visualization*. 225–228.
- Mark J. Kilgard and Jeff Bolz. 2012. GPU-accelerated Path Rendering. *ACM Trans. Graph.* 31, 6, Article 172 (Nov. 2012), 10 pages.
- R. Victor Klassen. 1991. Integer Forward Differencing of Cubic Polynomials: Analysis and Algorithms. *ACM Trans. Graph.* 10, 2 (April 1991), 152–181.
- Jerome Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. 2001. Real-time Fur over Arbitrary Surfaces. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics (I3D '01)*. 227–232.
- Zicheng Liao, H. Hoppe, D. Forsyth, and Yizhou Yu. 2012. A Subdivision-Based Representation for Vector Image Editing. *IEEE Trans. on Vis. and Computer Graph.* 18, 11 (2012), 1858–1867.
- Charles Loop and Jim Blinn. 2005. Resolution Independent Curve Rendering Using Programmable Graphics Hardware. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. 1000–1009.
- Jonathan T. Moon, Bruce Walter, and Steve Marschner. 2008. Efficient Multiple Scattering in Hair Using Spherical Harmonics. In *ACM SIGGRAPH 2008 Papers (SIGGRAPH '08)*. Article 31, 7 pages.
- David E. Muller. 1956. A Method for Solving Algebraic Equations Using an Automatic Computer. *Math. Tables Aids Comput.* 10 (1956), 208–215.
- Koji Nakamaru, Toru Matsuoka, and Masahiro Fujita. 2012. Distance Aware Ray Tracing for Curves. In *ACM SIGGRAPH 2012 Posters (SIGGRAPH '12)*. Article 103, 1 pages.
- Koji Nakamaru and Yoshio Ohno. 2002. Ray Tracing for Curves Primitive. In *WSCG*. 311–316.
- Diego Nehab and Hugues Hoppe. 2008. Random-access Rendering of General Vector Graphics. In *ACM SIGGRAPH Asia 2008 Papers (SIGGRAPH Asia '08)*. Article 135, 10 pages.
- Diego Nehab and Hugues Hoppe. 2012. A Fresh Look at Generalized Sampling. *Foundations and Trends in Computer Graphics and Vision* 8, 1 (2012), 1–84.
- Jiawei Ou, Feng Xie, Parashar Krishnamachari, and Fabio Pellacini. 2012. ISHair: Importance Sampling for Hair Scattering. *Comput. Graph. Forum* 31, 4 (June 2012), 1537–1545.

- Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Hao Qin, Menglei Chai, Qiming Hou, Zhong Ren, and Kun Zhou. 2014. Cone Tracing for Furry Object Rendering. *IEEE Trans. Vis. Comput. Graph.* 20 (2014), 1178–1188.
- Zheng Qin, Michael D. McCool, and Craig Kaplan. 2008. Precise Vector Textures for Real-time 3D Rendering. In *Proc. of the 2008 Symposium on Interactive 3D Graphics and Games (I3D '08)*. 199–206.
- Nicolas Ray, Xavier Cavin, and Bruno Lévy. 2005. *Vector Texture Maps on the GPU*. Technical Report. Laboratoire Lorrain de Recherche en Informatique et ses Applications.
- Zhong Ren, Kun Zhou, Tengfei Li, Wei Hua, and Baining Guo. 2010. Interactive Hair Rendering Under Environment Lighting. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. Article 55, 8 pages.
- Alexander Reshetov and David Luebke. 2016. Infinite Resolution Textures. In *Proceedings of High Performance Graphics (HPG '16)*. 139–150.
- C.J.F. Ridders. 1979. A new algorithm for computing a single root of a real continuous function. *IEEE Transactions on Circuits and Systems* 26, 11 (1979), 979–980.
- T. W. Sederberg and T. Nishita. 1990. Curve Intersection Using BeÁzier Clipping. *Comput. Aided Des.* 22, 9 (Nov. 1990), 538–549.
- Pradeep Sen. 2004. Silhouette Maps for Improved Texture Magnification. In *Proc. of the ACM SIGGRAPH/Eurographics Conf. on Graphics Hardware (HWWS '04)*. 65–73.
- Erik Sirtorn and Ulf Assarsson. 2009. Hair Self Shadowing and Transparency Depth Ordering Using Occupancy Maps. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. 67–74.
- Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. 2012. Diffusion Curve Textures for Resolution Independent Texture Mapping. *ACM Trans. Graph.* 31, 4, Article 74 (July 2012), 9 pages.
- Marco Tarini and Paolo Cignoni. 2005. Pinchmaps: Textures with Customizable Discontinuities. *Comput. Graph. Forum* 24, 3 (2005), 557–568.
- Wikipedia. 2016. Root-finding Algorithms — Wikipedia, The Free Encyclopedia. (2016). https://en.wikipedia.org/wiki/Category:Root-finding_algorithms
- Sven Woop, Carsten Benthin, Ingo Wald, Gregory S Johnson, and Eric Tabellion. 2014. Exploiting Local Orientation Similarity for Efficient Ray Traversal of Hair and Fur. In *High Performance Graphics*. 41–49.
- Kui Wu and Cem Yuksel. 2017. Real-time Fiber-level Cloth Rendering. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '17)*. Article 5, 8 pages.
- Ling-Qi Yan, Chi-Wei Tseng, Henrik Wann Jensen, and Ravi Ramamoorthi. 2015. Physically-accurate Fur Reflectance: Modeling, Measurement and Rendering. *ACM Trans. Graph.* 34, 6, Article 185 (Oct. 2015), 13 pages.
- Xuan Yu, Jason C. Yang, Justin Hensley, Takahiro Harada, and Jingyi Yu. 2012. A Framework for Rendering Complex Scattering Effects on Hair. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '12)*. 111–118.
- Arno Zinke, Cem Yuksel, Andreas Weber, and John Keyser. 2008. Dual Scattering Approximation for Fast Multiple Scattering in Hair. *ACM Trans. Graph.* 27, 3, Article 32 (Aug. 2008), 10 pages.